

— E-BOOK · THE COMPLETE GUIDE

How to Operate an **AI Factory.**

A practical guide to turning GPU capacity into a governed, measurable, production-grade AI service — the frameworks, the economics, and the operating model.

12 chapters

4 frameworks

94 capabilities

CONTENTS

What this guide covers

This guide is written for the people who have to make an AI estate work in production — platform engineering, infrastructure, FinOps, and the executives funding it. It assumes you already have GPUs. The question it answers is how to operate them.

01	Why AI infrastructure broke the old playbook	03	02	The six frictions of an un-governed estate	04
	The shift from servers to a production environment			Idle GPUs, contention, fragmentation, blind costs	
03	What a control plane actually is	05	04	Framework 1 — the operating loop	06
	The factory analogy and where Vektor sits			See, Decide, Act, Govern, Measure, Port	
05	Placement: the highest-leverage decision	07	06	Framework 2 — the economics of a token	08
	Priority, SLA, residency, and cost objectives			What really sits inside cost-to-serve	
07	Multi-tenancy without silos	09	08	Observability for AI, not just infrastructure	10
	Entitlements, quotas, isolation, chargeback			Models, prompts, drift, guardrails, agents	
09	Framework 3 — the maturity model	11	10	The capability surface	12
	Five levels from ad-hoc to monetised			94 capabilities across 15 categories	
11	Framework 4 — the delivery path	13	12	A worked example, and what to ask vendors	14
	Design, Build, Operate, Monetize			Three teams, one estate, and a buyer checklist	

FOUR FRAMEWORKS YOU CAN REUSE

- **The operating loop** — the six functions any AI factory has to perform, in order.
- **The economics of a token** — the six cost components behind cost-to-serve.
- **The maturity model** — five levels, with the constraint that defines each.
- **The delivery path** — Design, Build, Operate, Monetize.

01 CHAPTER ONE

Why AI infrastructure broke the old playbook

For thirty years, infrastructure management assumed a particular shape of workload: many small, stateless, interchangeable units of compute, scheduled loosely, and measured in aggregate. AI breaks every one of those assumptions.

An AI estate runs a handful of very large, very expensive, highly stateful workloads whose requirements contradict each other. A training run wants a large block of GPUs for days and does not care about latency. An inference endpoint wants a small slice of GPU permanently and cares about nothing except latency. A batch enrichment job wants whatever is cheapest and can wait. Put all three on one estate without a scheduler that understands the difference, and the training job wins — because it asked first and never lets go.

The unit of value moved

In classic infrastructure, the unit you bought and the unit you sold were the same: a server, a VM, a core-hour. In an AI factory they are different. You buy GPU-hours. You deliver tokens. Everything interesting — utilisation, margin, cost-to-serve, whether a customer is profitable — lives in the conversion between those two units, and almost no traditional tooling measures it.

You buy GPU-hours. You sell tokens. The entire operating discipline lives in the gap between them.

Scarcity changes behaviour

Cloud infrastructure trained a generation of engineers to treat capacity as infinite and cost as someone else's problem. GPUs reversed that overnight. Capacity is genuinely finite, lead times are long, and the marginal cost of a careless decision is visible in the invoice. That makes allocation political: when capacity is scarce and allocation is informal, the loudest team wins rather than the most valuable workload. A control plane is, among other things, a way of taking that decision out of the corridor and putting it into policy.

WHAT THIS CHAPTER MEANS IN PRACTICE

- Workload **type** matters more than workload size — priority must be a property the platform enforces.
- Measurement must span **GPU-hours to tokens**, or the economics stay invisible.
- Allocation is a **governance** problem before it is a technical one.

02 CHAPTER TWO

The six frictions of an un-governed estate

Organisations rarely discover these one at a time. They arrive together, usually in the quarter after the first production AI workload goes live and the second one is proposed.

FRICTION	HOW IT SHOWS UP	WHAT IT ACTUALLY COSTS
Idle and stranded GPUs	Capacity unused between jobs, or fragmented into slices too small to schedule	Depreciation on hardware doing no work, plus premature purchase of more
Workload contention	Training runs degrading latency-bound inference	SLA breaches on the workloads that face customers
Fragmented operations	GPUs, fabric, runtimes, identity, and billing managed in separate tools	Slow incident response; engineers doing integration instead of delivery
Blind economics	One infrastructure invoice, no attribution by model, tenant, or workload	No pricing basis, no chargeback, no defensible business case
Manual governance	Access and audit evidence assembled by hand across systems	Weeks of effort per audit; risk of findings
Runtime lock-in	Serving coupled to one stack or provider	Inability to adopt faster runtimes; weak negotiating position

The compounding problem

These frictions are not independent. Blind economics prevents you from justifying the investment that would fix idle capacity. Fragmented operations make governance manual, which makes audits expensive, which makes the security team reluctant to approve the next workload. Contention pushes teams to request dedicated capacity, which fragments the estate further and drives utilisation down again. Each friction makes the others worse, which is why incremental tooling rarely resolves them and why the fix is usually structural.

Buying more GPUs is the most expensive way to solve a scheduling problem — and the most common.

03 CHAPTER THREE

What a control plane **actually** is

A control plane is not a monitoring dashboard, a scheduler, or a billing system. It is the layer that makes those things agree with each other — one place where capacity, policy, workload, and cost are described in the same terms.

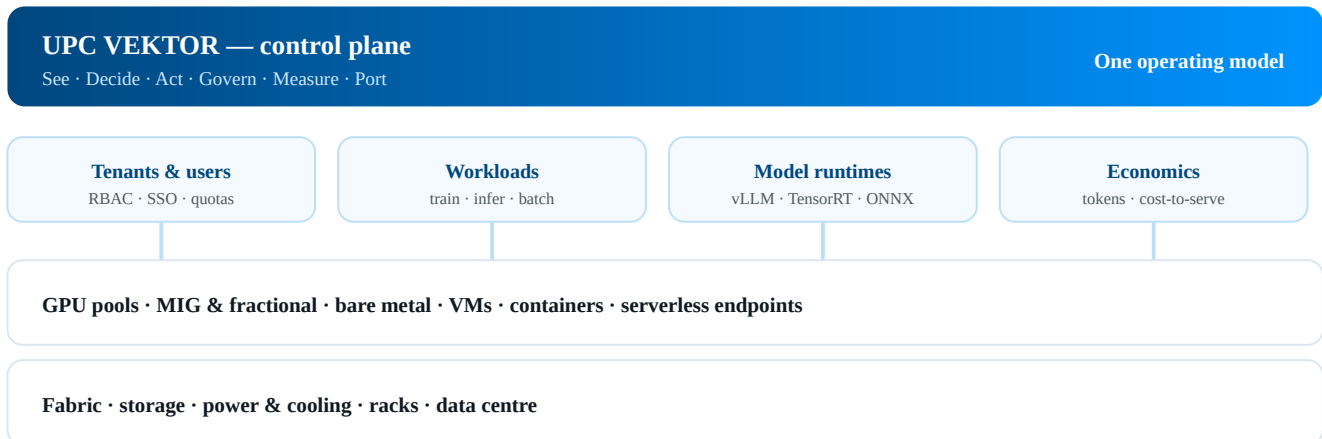


FIGURE 1 — WHERE THE CONTROL PLANE SITS

The factory analogy

If the AI estate is a production facility, the GPUs are the machines, servers and networking are the production floor, models and workloads are the products, and data is the raw material. UPC Vektor is the brain and control system that coordinates the factory. It does not replace the GPUs and it does not perform the model's reasoning — it manages the infrastructure and workloads so that enterprise AI operates reliably and efficiently.

WHAT THE CONTROL PLANE CONNECTS

- **Infrastructure and GPU capacity** — inventory, pools, health, utilisation, fabric topology.
- **Workloads and runtimes** — training, inference, batch, experimentation, and the serving stacks under them.
- **Tenants and policy** — identity, entitlements, quotas, RBAC, audit, and SLA commitments.
- **Economics** — usage metering, token throughput, cost-to-serve, showback, chargeback, and revenue.

The test of whether something is genuinely a control plane is simple: can it make a decision? A dashboard tells you a GPU is idle. A control plane reclaims it. A monitoring tool tells you latency degraded. A control plane knows which workload caused it, which policy was violated, and what to pre-empt.

04 FRAMEWORK ONE

The operating loop

Every AI factory performs six functions, whether or not anyone has named them. When they are performed by separate tools and different teams, the loop leaks. When one layer performs all six, each step informs the next.

The operating loop

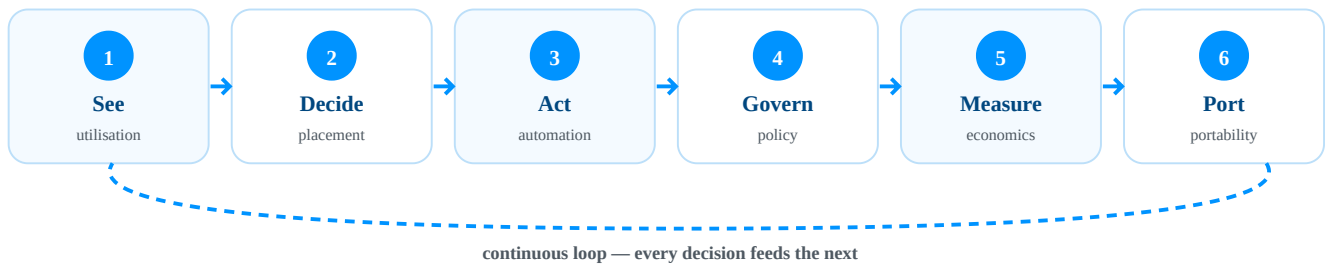


FIGURE 2 — THE SIX-STEP OPERATING LOOP

STEP	THE QUESTION IT ANSWERS	WHAT GOOD LOOKS LIKE
1 • See	What is actually happening across the estate?	GPU utilisation, health, workload status, fabric performance, tenant consumption, and cost in one view
2 • Decide	Where should this workload run?	Placement by availability, priority, latency need, SLA, policy, entitlement, and cost objective
3 • Act	Can the platform do it without a human?	Provisioning, deployment, serving management, runbooks, remediation, rollback, ChatOps
4 • Govern	Who may do what, under which policy?	Tenant onboarding, RBAC, SSO, MFA, API access, audit logging, confidential computing
5 • Measure	Is this running economically?	GPU and token consumption, latency, cost-to-serve, showback and chargeback by tenant and model
6 • Port	Can we move this if the economics change?	Runtime abstraction across vLLM, LiteLLM, TensorRT-LLM, ONNX, CUDA, and public APIs

A loop with a missing step is not a loop. It is a handoff — and handoffs are where AI estates lose both money and time.

Placement: the highest-leverage decision you make

Of the six steps, placement returns the most for the least effort. It is also the one most often left to whoever submitted the job first.

Placement is a policy expression, not a queue

A naive scheduler answers one question: is there a free GPU? A placement engine answers seven. Is there capacity of the right shape? What priority does this workload carry? Does it have a latency commitment attached? Is there an SLA behind it? Does policy restrict where it may run? Does the tenant have the entitlement? And what is the cost objective — fastest, cheapest, or most efficient?

The difference is visible immediately in one scenario every operator recognises: a long training run and a customer-facing inference endpoint arrive at the same estate. Without placement policy, the training job takes the capacity because it asked for more of it. With placement policy, the inference endpoint holds guaranteed capacity, the training job runs against what remains, and is pre-empted when the endpoint needs to scale.

PLACEMENT INPUTS WORTH ENCODING

- **Workload class** — interactive inference, batch inference, training, fine-tuning, experimentation.
- **Commitment** — whether an SLA or a latency target is attached, and to whom.
- **Residency and policy** — regions, sovereignty clauses, confidential-computing requirements.
- **Entitlement** — what the tenant is contractually allowed to consume.
- **Cost objective** — whether the workload optimises for speed, price, or utilisation.

Fractional allocation changes the arithmetic

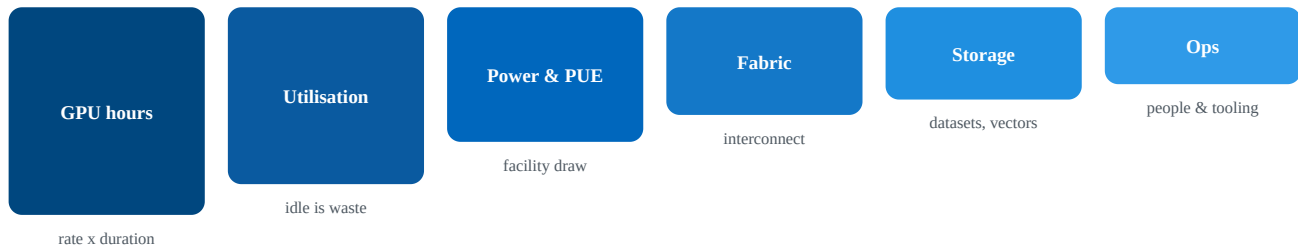
Much of what looks like a capacity shortage is a granularity problem. A workload needing a fraction of an accelerator will otherwise occupy a whole one. MIG, time-slicing, and logical partitioning let a single GPU serve several small workloads, which is often the fastest route to higher utilisation without buying anything — provided the control plane can see the slices, schedule against them, and meter them separately.

06 FRAMEWORK TWO

The economics of a token

Cost per million tokens is the closest thing an AI factory has to a unit cost. Most organisations calculate it from GPU rental alone, which is why their private-versus-public comparisons are usually wrong.

What actually sits inside cost per million tokens



Most teams meter only the first bar. The control plane meters all six — which is why private-versus-public comparisons usually change once the full stack is measured.

FIGURE 3 — THE SIX COMPONENTS OF COST-TO-SERVE

Why the full stack matters

Two estates with identical GPUs can differ by a wide margin in cost per million tokens, and the difference rarely comes from the hardware. It comes from utilisation — whether those GPUs are busy — and from the facility and fabric costs carried whether or not they are. An estate at 30% utilisation pays for 100% of its power, cooling, interconnect, and depreciation. Raising utilisation is mathematically identical to cutting unit cost, and it requires no procurement.

QUESTION	WHAT YOU NEED TO ANSWER IT
What does it cost to train this model?	GPU-hours by job, attributed to a project and programme
What is the cost per inference or per token?	Token throughput metered per model and endpoint, divided into fully loaded cost
Which business unit uses the most capacity?	Consumption metered per tenant, with entitlements to compare against
How much does idle capacity cost us?	Utilisation over time against fixed facility and depreciation cost
Which customers are profitable?	Cost-to-serve per tenant set against plan revenue and margin
Private or public for this workload?	Fully loaded private cost-to-serve versus public API pricing, per model

Utilisation is not an infrastructure metric. It is the largest single lever on unit cost, and it is free.

07 CHAPTER SEVEN

Multi-tenancy without silos

The instinctive response to contention is separation: give each team its own hardware. It resolves the argument and destroys the economics, because every silo carries its own idle capacity.

One estate, many tenants, enforced boundaries



FIGURE 4 — MULTI-TENANCY WITHOUT DEDICATED SILOS

Logical isolation, shared substrate

The alternative is to share the estate but make the boundaries real — enforced by policy rather than by physical separation. Each tenant receives entitlements describing what it may consume, quotas capping it, access rules governing who acts on its behalf, and metering that attributes consumption back. From the tenant's perspective the estate behaves as if dedicated. From the operator's perspective it is one pool running at a utilisation no silo could reach.

WHAT A TENANT BOUNDARY HAS TO CARRY

- **Entitlement** — GPUs, tokens, endpoints, and services the tenant may consume.
- **Quota** — the ceiling, enforced at the platform rather than by convention.
- **Priority class** — whether this tenant's work pre-empts, or is pre-empted.
- **Access and audit** — identity, roles, API keys, and an inspectable trail.
- **Economics** — metered consumption producing showback, chargeback, or an invoice.

The internal service-provider model

Once those five elements exist, an internal platform team can operate the way a service provider does: publish a catalogue, onboard tenants against plans, enforce SLAs, and report cost and margin per tenant. That shift — from cost centre to internal service provider — is usually what turns AI infrastructure from something finance tolerates into something it can plan around.

08 CHAPTER EIGHT

Observability for AI, not just infrastructure

Infrastructure observability tells you the GPU is healthy. It will not tell you the model has started hallucinating, that a retrieval pipeline is returning stale context, or that an agent is looping and burning tokens.

Two layers, both required

An AI factory needs the classical layer — infrastructure, network, database, virtualisation, device and interface monitoring, metrics, health and SLA analytics — and an AI-native layer above it. The second layer watches things the first cannot see, and most incidents that reach a customer originate there.

Model and LLM monitoring

Model behaviour, LLM health, inference performance, and model-level telemetry — the equivalent of APM for a model.

Prompt and trace monitoring

Prompt, request, response, and distributed traces across an AI transaction, so a bad answer can be reconstructed.

Token and latency analytics

Token consumption, throughput, endpoint latency, and the trend between them — where performance meets cost.

Drift and quality

Model drift, data drift, hallucination, and quality degradation — failures that never trigger an infrastructure alert.

Safety and guardrails

Safety incidents, guardrail activity, and policy violations, recorded as events rather than discovered later.

Agents, RAG, tools, vectors

Agent behaviour, retrieval pipelines, tool execution, and vector database telemetry — the parts that fail silently.

A healthy GPU serving a drifting model at full speed is the most expensive failure mode in an AI factory.

The practical argument for putting both layers in one control plane is correlation. When latency rises, the question is whether the cause is a saturated GPU, a fabric bottleneck, a slow retrieval step, or a model change. Answering that across four tools takes hours; answering it in one takes a query.

09 FRAMEWORK THREE

The AI factory maturity model

Most organisations can place themselves on this ladder in under a minute. The useful part is not the label but the constraint — each level is defined by the specific thing that stops it moving up.

1	Ad-hoc	GPUs exist; access is negotiated informally. No inventory, no entitlements, no attribution. Constraint: nobody knows what is actually running.
2	Visible	Consolidated inventory and utilisation monitoring across the estate. Idle capacity becomes apparent. Constraint: seeing waste without the means to reclaim it.
3	Governed	Tenants, entitlements, quotas, RBAC, and audit in place. Placement follows policy rather than arrival order. Constraint: costs still unattributed, so investment is hard to justify.
4	Measured	Consumption metered per tenant, model, and workload; cost-to-serve and token economics visible. Constraint: data exists but does not yet drive placement or pricing.
5	Monetised	Economics feed decisions automatically — placement optimises against real cost, chargeback runs, margin is reported per tenant, and workloads move between private and public on economics. Constraint: none structural; the estate now scales as a business.

How to use it

Two rules make this model practical. First, levels are not skippable: metering an estate you cannot govern produces numbers nobody can act on, and governing an estate you cannot see produces policy nobody can enforce. Second, the jump from three to four is where most programmes stall — governance is visibly valuable to security, while metering is only valuable once someone is accountable for the number it produces. Naming that owner before you build the dashboards is usually what determines whether level four sticks.

A QUICK SELF-ASSESSMENT

- Can you state current GPU utilisation across the whole estate, right now, without asking anyone? **(Level 2)**
- Would a training job submitted this afternoon be able to degrade a customer-facing endpoint? **(Level 3)**
- Can you produce cost-to-serve for a named model, this month, by tenant? **(Level 4)**
- Has a workload moved between private and public capacity because the economics said so? **(Level 5)**

10 CHAPTER TEN

The capability surface

Operating an AI factory touches far more than GPUs. UPC Vektor consolidates 94 capabilities across 15 categories — from physical data-centre management through AIOps and GPU orchestration to AI observability, FinOps, and governance.



11 FRAMEWORK FOUR

The delivery path

Capability alone does not produce an operating factory. UPC Vektor structures the work across four stages — from infrastructure design to operational handover — giving a production-ready factory with full autonomy after delivery.

01 · Design

Plan GPU topology, tenant architecture, network fabric, service catalog, capacity, and token pricing for the factory.

02 · Build

Deploy the infrastructure stack, onboard tenants, activate workloads, connect LLM APIs, and validate token flows.

03 · Operate

Monitor GPU health and token throughput, optimise workloads, manage costs, and enforce governance across production.

04 · Monetize

Meter tenant usage, automate chargeback and showback, and measure revenue against real-time cost-to-serve.

What to settle in Design

The decisions that are expensive to reverse are nearly all taken in the first stage. Tenant architecture determines whether business lines can ever be charged separately. GPU topology and fabric determine which workload shapes are viable. The service catalogue determines what teams can request without a conversation. And the pricing model — even a notional internal one — determines whether anyone will be accountable for consumption later. A factory that reaches Operate without those settled tends to arrive at level three of the maturity model and stop.

QUESTIONS WORTH ANSWERING BEFORE BUILD

- Which tenants exist on day one, and what entitlement does each hold?
- Which workload classes carry SLAs, and what may pre-empt what?
- What must stay in-region or inside a confidential-computing boundary?
- Who owns the cost-to-serve number once metering is live?
- Which runtimes must be supported, and which are we deliberately abstracting?

12 CHAPTER TWELVE

A worked example, and what to ask

A private AI platform serves three groups: research training a large language model, customer service running a real-time assistant, and finance processing documents in scheduled batches.

GROUP	WITHOUT A CONTROL PLANE	WITH UPC VEKTOR
Customer service	Latency degrades whenever research submits a large job	Holds the priority and capacity required for low-latency service
Research	Consumes most of the estate because nothing stops it	Uses appropriate GPU pools for long training, pre-empted by policy
Finance batch	Runs during expensive peak windows by accident	Scheduled onto available or lower-cost capacity
The platform team	Investigates issues layer by layer across tools	Monitors centrally, automates remediation, reclaims idle capacity
Finance	Sees one invoice, attributes nothing	Token consumption and cost attributed to each team

The outcome is not only better utilisation. It is a more predictable AI service for every team on the platform — which is the thing that lets a fourth and fifth workload be approved.

A buyer's checklist

Whatever platform you evaluate, these questions separate a control plane from a dashboard:

- | | |
|---|---|
| ☐ Can it enforce priority, or only report contention? | ☐ Does it meter tokens, or only GPU-hours? |
| ☐ Can it reclaim idle and stranded capacity automatically? | ☐ Are tenant entitlements enforced by the platform? |
| ☐ Does it produce chargeback finance will accept? | ☐ Can it schedule fractional GPU slices, not just whole GPUs? |
| ☐ Does it cover AI-native signals — drift, guardrails, agents? | ☐ Can workloads move runtimes without application changes? |
| ☐ Is audit evidence inspectable without a manual exercise? | ☐ Does it compare private and public cost per model? |
| ☐ Can it pre-empt a job, and roll a deployment back? | ☐ Does one view cover fabric, power, and thermal as well as GPU? |

If the answer to most of these is "it shows you," not "it does it," you are looking at observability — not a control plane.

IN CLOSING

From GPU capacity to AI business value

An AI factory is not defined by how many GPUs it contains. It is defined by whether the organisation can answer, on any given day, what is running, who authorised it, how well it is performing, and what it costs.

Those four questions map precisely onto the frameworks in this guide. The operating loop makes them answerable. Token economics makes the last one credible. The maturity model tells you which one you cannot yet answer. And the delivery path is how organisations get from owning capacity to operating a service.

THE FIVE THINGS WORTH TAKING AWAY

- **Priority is a platform property.** If policy is not enforced by the system, it is not policy.
- **Utilisation is the cheapest lever** on unit cost, and requires no procurement.
- **Share the estate, enforce the boundaries.** Silos resolve arguments and destroy economics.
- **Measure GPU-hours to tokens.** Anything less leaves the business case unprovable.
- **Portability is leverage** — commercially as much as technically.

BOTTOM LINE

Turn GPU capacity into AI revenue — run the entire factory with intelligence.

Want this assessed against **your estate**?

Book a walkthrough of UPC Vektor, or ask us to run a utilisation and cost-to-serve assessment.

UPC VEKTOR™
UNITEDLAYER.COM



EMAIL
contact@unitedlayer.com



PHONE
[+1-888-853-7733](tel:+1-888-853-7733)



X
x.com/unitedlayer



LINKEDIN
linkedin.com/company/unitedlayer